2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Control Architecture for Robot Cells to Enable Plug'n'Produce

Martin Naumann, Kai Wegener, and Rolf Dieter Schraft, Fraunhofer IPA, Germany.

Abstract - **This paper deals with the concept of a control architecture for robot cells that enables Plug'n'Produce according to Plug'n'Play in the office world. To achieve this, a software module called "Interconnector Module" takes as input descriptions of devices and processes. These descriptions are then automatically evaluated in order to offer the user commands to use the functionality of the robot cell in its current setup. The evaluation consists of several steps that are processed in different sub-modules of the Interconnector Module. In this paper the concept of this Interconnector Module is introduced.**

I. Introduction

The main field of application for robots today is mass production [1]. The tasks robots have to fulfil in mass production are mostly highly repetitive and do not change over an extended period of time. Therefore, the main requirements for robots used in mass production are short cycle times. The goal of *SMErobot*™ [2] is to broaden the field of applications for robots from mass production to small lot size production, as it is typically encountered in small and medium sized enterprises (SMEs). Because of small lot sizes, fast adaptability of robot and surrounding cell to new products and processes is much more important for SMEs than short cycle times. To make this possible the programming of applications for robot cells and the integration of new devices into these robot cells has to be much easier than today.

II. Approach and Scope of this Paper

In the office world it is very easy to install and use new devices. For example, to install a printer to your PC, you just plug it in. The entire configuration is then done automatically and your application will offer you the service "print". This automatic configuration is called "Plug'n'Play". Carried forward to a production environment this would mean that you would connect e.g. a robot to a cell controller and it would offer you the command "`move_to`" on a HMI. Even more advanced, it could mean that you connect e.g. a robot and a gripper to a cell controller and the cell controller would recognize the new possibilities enabled through the combination of two or more devices and offer you the command "`pick and place`". To achieve this, the cell controller needs to know about the functionality of the connected devices and must be able to draw conclusions which services it can offer to a user. The approach pursued in this paper is based on device descriptions evaluated by an Interconnector module in order to offer commands representing the functionality of the robot cell to a user.

The ability - provided by the Interconnector Module - to add devices to a robot cell and to use the functionality of these devices without the need of configuration is called "Plug'n'Produce", according to "Plug'n'Play" in the office world. Plug'n'Produce (P'n'P) can be broken down into several layers depending on the amount of configuration that is done automatically. These layers can be seen in figure 1.

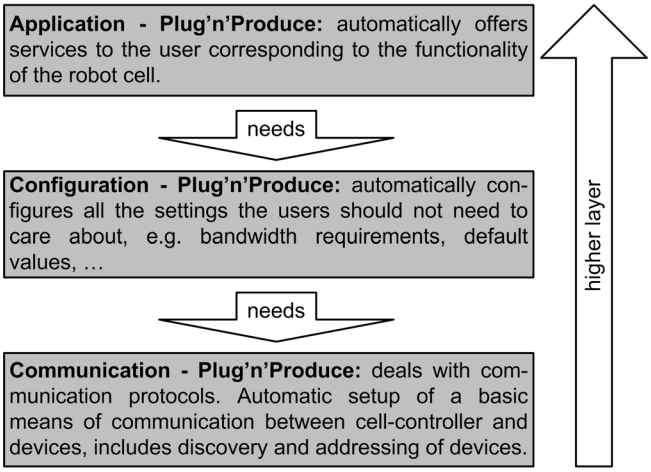

Figure 1: Plug'n'Produce layers

This paper introduces a concept to offer the user of a robot cell in SME-environments an easy means of programming a cell without the need to care about communication and configuration. Therefore, the focus of this paper lies on the Application-P'n'P-layer. Of course, this layer depends on the Configuration- and the Communication-P'n'P-layers in order to get to know which devices are available, to communicate with these devices and to get to know the descriptions of these devices [3]. However, the realization of the two lower layers will not be within the scope of this paper.

III. State of the Art

State of the art of describing device categories with certain, common functionalities are device profiles that exist for different protocols like EDDL [4], XIRP [5] or UPnP [6]. These device profiles define communication interfaces that have to be supported by a device in order to belong to a certain device category. The functionality of the device can

This work has been funded by the European Commission's Sixth Framework Program under grant no. 011838 as part of the Integrated Project SMErobot™.

partly be inferred from the communication interface, but it is not itself part of a device profile. Therefore, device profiles do not contain enough information to allow detailed assumptions about the differences/similarities of the functionalities of devices.

In the domain of knowledge representation, languages have been developed that can be used to describe functionalities of devices in form of a taxonomy plus additional attributes. The most popular of these languages, OWL (Web Ontology language) [7], was developed as a key technology of the Semantic Web [8], trying to add meaning to the information that is today merely displayed in the internet.

In the context of home entertainment systems, a function planning module was developed within the SmartKom project [9]. This module tries to serve complex user requests by first determining which devices are necessary and then determining how to control devices based on abstract descriptions of the functionalities of devices [9].

In this paper, the concept of device descriptions augmented by a component describing the device's functionality with the help of a knowledge representation language will be used to infer and use the functionality of a robot cell according to the function planning module of the SmartKom project thus enabling Application-P'n'P.

IV. Application-P'n'P: Overview

Application-P'n'P as the highest P'n'P-layer has the goal to offer the user as easy as possible means of using the functionality of the connected devices. In the context of *SMErobot*™ this means offering the user as easy as possible means of programming robot cells.

State of the art of programming robot cells is to enter commands in the dialog of some sort of a programming system. The entered commands are then uniquely mapped to the according devices. This is an appropriate way of programming as long as the user has detailed knowledge about the control structure of devices as well as about programming itself. In the context of a *SMErobot*™-application this cannot be granted. Users of robot cells in SME-environments normally know a lot about the processes they have to perform in order to achieve their desired result, but have only minor knowledge about programming devices (a robot is a special kind of device). Therefore, the programming of robot cells in SME-environments should be possible without the need of device programming. Instead, programming should be focused on the processes the user wants to execute. In this paper, this will be called "**process-oriented programming**" and the corresponding commands will be called "**process commands**" as opposed to traditional "device commands". Process commands trigger whole processes like drilling a hole or gripping a part, while device commands trigger a state change in a single device like setting a digital output or moving a robot from point A to point B. Figure 2 illustrates the differences.

Process command	Device commands

`DrillHole(x=50,y=90,d=30)`	`MoveTo(50, 90, 70);`
	`SetPort(20);`
	`MoveTo(50, 90, 40);`
	`MoveTo(50, 90, 70);`
	`ResetPort(20);`

Figure 2: Process command vs. device commands

Process-oriented programming imposes the following new requirements on the programming system:

- No manual device integration effort (→ Communication- and Configuration-P'n'P)
- Process commands as user input

Therefore, instead of just mapping device commands to the according devices, the tasks of the programming system are much more complex:

- Automatic Setup of communication with devices (→ Communication-P'n'P)
- Automatic Configuration of devices (→Configuration-P'n'P)
- Evaluation of the functionality of single devices
- Inference of the functionality of the robot cell as a whole
- Supplying of process commands corresponding to the robot cell functionality to a HMI
- Generation of device command sequences to execute the offered process commands

To fulfill these tasks the "Interconnector Module" is introduced. The functionality and elements of this Interconnector Module will be described in detail in the following chapters. Figure 3 compares the concept of the Interconnector module with a traditional programming system. It can be seen that in a traditional programming system the user has to interact directly with the devices whereas in a system containing the Interconnector Module, this Interconnecter Module acts as an intermediate and can transform device commands into more intuitive process commands.

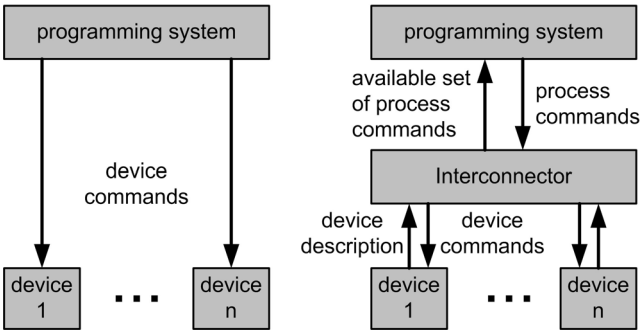

Figure 3: Traditional programming system vs. programming system for process-oriented programming

V. Elements of Application-P'n'P

In order to do its job the Interconnector Module must be supplied with the following **information**:

- Device descriptions that contain information about the available devices
- A library of process descriptions that contain information about processes that can possibly be executed by the robot cell.
- An ontology to define and relate the terms of the above mentioned descriptions.

To evaluate this information the following **methods** are required:

- A method to evaluate possible combinations of devices (e.g. combination of a robot and a gripper mounted to its flange)
- A method to generate device descriptions for combined devices out of the device descriptions of single devices
- A method to determine the processes that can be executed by the robot cell
- A method to generate sequences of device commands for all processes executable by the robot cell

Figure 4 shows the workflow of the Interconnector Module. Device descriptions are supplied by the (lower) Communication- and Configuration-P'n'P-layers. In a first step, possible device combinations are determined. Next, device descriptions for these combinations are generated. These device descriptions are compared to process descriptions to determine all executable processes. These executable processes are offered to a user on a HMI. The user defines a sequence of processes in order to fulfill a certain task, e.g. making part of a shelf out of a board. Out of this process sequence, a device command sequence is generated that can finally be executed by the robot cell.

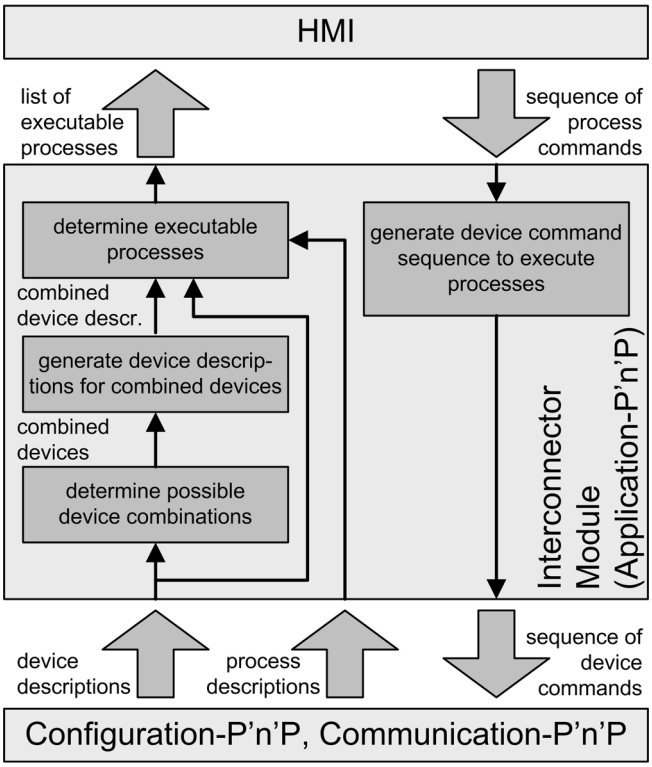

Figure 4: Workflow of the Interconnector Module

VI. Descriptions

As shown above, two types of descriptions are necessary to achieve Application-P'n'P:

- **Device descriptions** contained in the memory of a device that are loaded into the cell controller when the device is integrated into the robot cell.
- **Process descriptions** out of a – possibly application specific – library of process descriptions stored in the cell controller.

Both description types are divided into sections describing different aspects of devices/processes. Some of these sections are mandatory, others are optional. Some can occur in device as well as in process descriptions, others are specific for either device or process descriptions.

In the following, these different sections will be introduced in detail.

A. Functional descriptions

Functional descriptions express the offered functionality of devices as well as the required functionality of processes in an abstract, symbolic way by introducing the concept of skills [11]. A skill represents a certain functionality of a device, e.g. a robot can move its flange, which is described by the skill "`MoveProgrammable`" and can attach another device like a drilling tool to its flange, which is described by the skill "`CanAttach`".

Description Logics languages [12] are made to express, access and reason about such kind of structured knowledge like the above introduced skills. One of the most popular Description Logics languages is OWL DL. The OWL language (Web Ontology Language) was developed as a major technology for the implementation of the Semantic Web (see chapter III). OWL DL is a sublanguage of OWL (Full) made especially for the structured representation of domain knowledge to automatically reason about this knowledge. With the help of OWL DL it is easy to describe concepts in the form of class-subclass relations. Classes may have properties to describe details and restrictions to define if a certain individual is a member of a certain class. Subclasses specialize their parent-classes by adding more restrictions.

By means of this language it is possible to express skills and their respective properties allowing abstract and general as well as detailed and specialized descriptions in a way that allows reasoning about the skills.

Functional descriptions are necessary to evaluate if a certain process can be executed by the current setup of the robot cell. This is done by matching the skills offered by the available devices (expressed in their device descriptions) with the skills required by a process (expressed in its process description).

Functional descriptions are a mandatory part of every device and process description.

B. Device control descriptions

Device control descriptions modell the control system of a device in the form of a state-chart. The state-chart can have as many states as necessary, but depending on the functional description of a device certain states are mandatory. If the functional description of a device contains a certain skill, the state-chart must contain the respective state(s), e.g. if the functional description of a drilling tool contains the skill "`CanRotate`", the device control description of this device must contain a state "`Rotating`". The transitions of the state-chart describe the commands that trigger the state changes.

There exist several languages to describe state-charts. One of them is SCXML [13]. SCXML allows the concurrent execution of parallel state-charts and their synchronization and is therefore well suited for the use in device control descriptions.

Device control descriptions are a mandatory part of device descriptions in order to evaluate the necessary sequence of commands to reach certain states – that means, to execute a certain task.

C. Process sequence descriptions

Process sequence descriptions have the purpose of describing the sequence of a certain process in an abstract way. "In an abstract way" means, that they must describe which states must be reached in which order to execute a certain process, but they must not describe how these states can be reached as this depends on the devices actually used. Therefore process sequence descriptions are the counterparts of device control descriptions. Device control descriptions express how certain states can be reached while process sequence descriptions express the required sequence of states in order to execute a certain task. Therefore, process sequence descriptions are expressed as SCXML-state-charts, too.

Process sequence descriptions are a mandatory part of process descriptions. The states contained in a state-chart must correspond to the skills contained in the functional description of the process description.

Figure 5 shows a state-chart describing a drilling process including a device with the skill "`CanMove`" and the corresponding state "`BeAtPos`" (→ a robot), a device with the skill "`CanRotate`" and the corresponding state "`Rotating`" (→ a drilling tool) and a device with the skill "`CanClamp`" and the corresponding state "`Clamped`" (→ a clamping device).

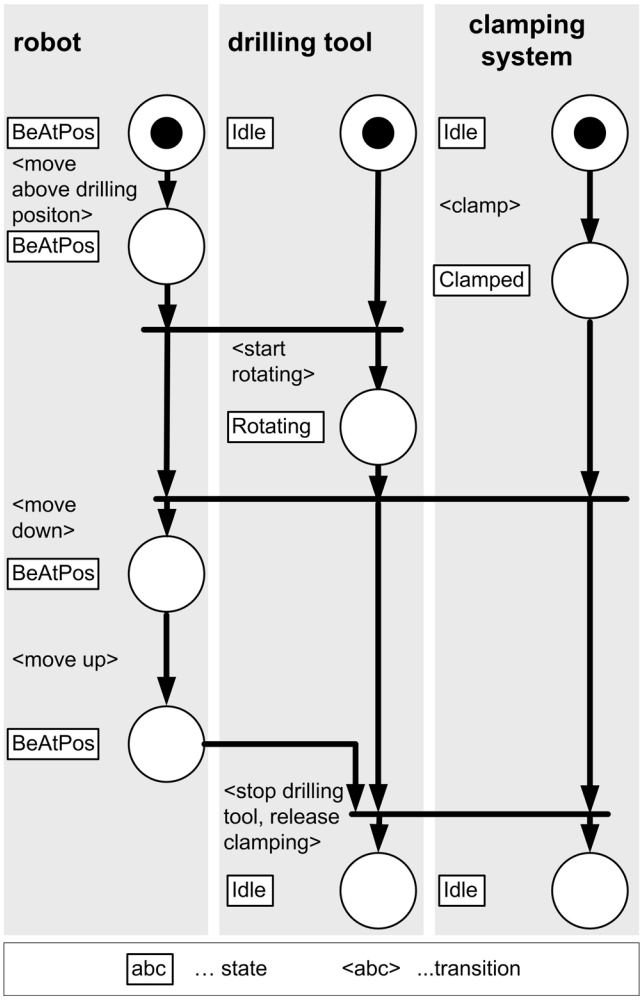

Figure 5: State-chart describing a drilling process

D. Optional descriptions

Apart from the above mentioned mandatory descriptions, further optional descriptions (with special purposes) are possible, e.g.:

- Geometrical descriptions: describe the body structure of a device. These descriptions could e.g. be used to simulate the robot cell or as input data for a path planner/collision avoidance algorithm.
- Kinematical descriptions: describe the kinematical chain of a device. This type of description is especially useful for robots in order to check the reachability of certain points.

In order to use these optional descriptions special evaluation functions would be necessary.

E. Code fragments embedded into descriptions

For some purposes it could be useful to integrate code fragments into device or process descriptions for special purposes. Examples for such code fragments could be functions integrated into a gripper description that take as input data CAD-models of an object and return possible gripping points as a result.

VII. Ontology

The descriptions from the previous chapter have the purpose of allowing the Interconnector Module to infer possible device combinations, to infer processes that can be executed and to generate device command sequences for executable processes. All this should be done without human intervention. Therefore, the Interconnector Module needs to "understand" the descriptions, meaning it needs to know about the concepts represented by the terms used in the description.

To achieve this, an ontology has to be introduced which defines all relevant concepts (called "classes") of the considered domain, in our case the robotic domain. Ontologies generally describe [14]:

- **Instances of classes**, e.g. "`robot_XY`" as an instance of the class "`Articulated_Robot`".
- **Classes**, e.g. "`Articulated_Robot`", "`Flange`", "`Command`", "`Skill`".
- **Attributes** of classes, e.g. "`weight`", "`color`".
- **Relations** between classes, e.g. "*`Articulated_Robot` is a subclass of `Robot`*" or "`every` *`Robot`* `has a` *`Flange`*".

Figure 6 shows the taxonomy of a set of classes representing skills. Not shown are the attributes of the skills, e.g. the class "`SkillFix`" has the attribute "`maxObjectWeight`" describing the maximum weight of the object that can be fixed. All three subclasses of "`SkillFix`" inherit that attribute, but can possess more specialized attributes like e.g. "`maxObjectDiam`" for "`SkillFixCylinder`".

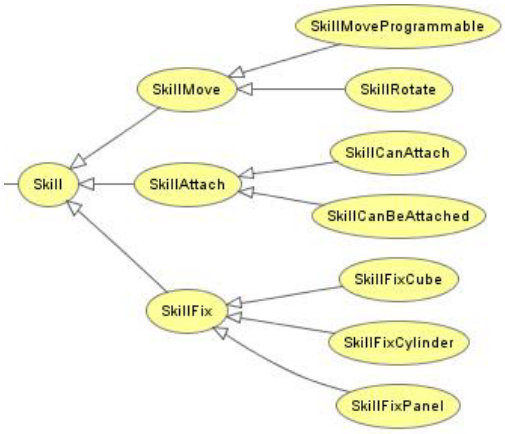

Figure 6: Part of the robot ontology

Example: The functional description of a concentric gripper contains an instance of "`SkillFixCylinder`" meaning that the concentric gripper can fix cylinders. The functional description of a pick-and-place process contains a device definition (in form of a class definition) stating that amongst others a device (with a functional description) containing an instance of "`SkillFix`" is required in order to execute the process. With the help of the ontology it can be inferred that the concentric gripper is suitable because it has a skill "`SkillFixCylinder`" which is a subclass of "`SkillFix`" and therefore suitable.

This simple example shows that all domain knowledge – as trivial as it might be for a human – has to be formally described (in an ontology) to allow a computer to make use of this knowledge in order to evaluate device descriptions.

The ontology is also the place where the relation between skills and states is defined (see chapter VI.B). This is done by the relation "requiredState" of a skill that defines states that are mandatory for certain skills, e.g. the skill "SkillRotate" requires a state "Rotate".

To define ontologies many languages have been developed. The most widespread one is OWL introduced in chapter VI.A in the context of functional descriptions. Therefore, it will be used for the definition of the robot domain ontology as well because the purpose of the ontology is to "understand" the functional descriptions.

VIII. Evaluation of Descriptions

The evaluation of device and process descriptions has two goals:

- to infer possible combinations of devices and
- to infer processes that can be executed by the robot cell.

A. Combination of devices

A combination of devices means that two or more devices are attached in order to execute a certain process. This could be necessary because one single device does not have the functionality required to execute this process, e.g. a robot alone can only move its flange and a drilling tool alone can only rotate a drill. But if the drilling tool is attached to the flange of the robot – if the devices are combined – they can drill a hole.

The combination of devices is a two step process. First, it has to be determined which devices can be combined. Second, a new device description specifying the functionality of the combined devices has to be generated.

1) Determining possible device combinations

Devices that can attach or can be attached to another device must have an interface for this purpose. This means, they must have the special functionality to attach to another device. This functionality can be represented by a skill, namely the skill "`SkillAttach`". This skill means that a device has an interface to attach another device. If two devices have this skill, it is possible to combine them if their interfaces match. Therefore, it is defined in the ontology that "`SkillAttach`" requires an attribute describing on the one side the interface the device possesses and on the other side the interface the device requires.

The interface of the device is described as an instance of the class "`DeviceInterface`" defined in the ontology. The interface the device requires is described as the

definition of a subclass of the class "`DeviceInterface`", specifying in detail the needed requirements. With the help of this interface descriptions it is possible to infer possible device combinations with a general purpose reasoner like e.g. Pellet [14] by checking if a "`DeviceInterface`" instance satisfies one or more "`DeviceInterface`" subclass definitions. If this is the case, these two devices can be combined.

2) Generating Device descriptions

As soon as it has been determined that two devices can be combined a new device description that treats these two devices as one single device has to be generated out of the device descriptions of the single devices. The integration of the device descriptions differs for functional descriptions and for all other kinds of descriptions:

- **Functional descriptions:** The skills of the functional descriptions of two devices that are combined are simply merged into one functional description. The only exception is the skill "`SkillAttach`" used to combine the devices. This skill is omitted in the new combined description as it is no longer available (because the devices are already combined).
- **All other descriptions:** all other description types are not merged but just copied into the new description.

B. Executable processes

The determination of executable processes out of a process library works similar to the combination of devices. Each functional description of a device contains a device instance including all the skills of the device. Each functional description of a process contains one or more device class definitions (for each device needed) specifying which skills are necessary. A general purpose reasoner can infer if a certain device instance satisfies one or more device class definitions. In this case, the corresponding device can be used to execute the process(es) containing this(these) device class definition(s).

IX. Generation of Device Command Sequences

The generation of device commands to execute a specific task is based on the state-chart of the process sequence description. The state-chart describes the states the involved devices have to reach, their order and synchronizations that must be taken into account. These states of the state-chart are mapped to states of the state-chart of the device control descriptions of the involved devices. The mapping is possible because the states of the process state-chart and the states of the device state-charts are related by the ontology. Once the state a device has to reach is known, a path from its current state to this goal state must be searched in the state-chart [16]. The commands (that represent the transition conditions of the state machine) along this path must be executed in order to reach the goal state. If this path-search is done for the whole process state-chart, the result is a sequence of device commands that has to be executed to fulfill the specific task.

X. Conclusion and Outlook

The presented concept of a robot cell control architecture allows easy programming of a robot cell and therefore permits users with little knowledge of (robot) programming to use robots in a SME-environment. The user still has to do some programming, but on the process-command level instead of using device commands.

The concept is based on device and process descriptions that are evaluated in an Interconnector Module to infer the functionality of the robot cell and offer the user corresponding process-commands. The accuracy of the inferred functionality strongly depends on the descriptions and the underlying ontology.

Therefore, further work has to be dedicated to particularize the specification of the device and process descriptions and the development and/or adaptation of the algorithms necessary for their evaluation. Simultaneously, more knowledge must be added to the robot ontology, as this ontology is the basis for the evaluation of the descriptions.

As a next step the concept presented in this paper will be realized for test applications (wood-working, pick and place). Thereby the strengths and weaknesses of the concept as well as possible improvements will become obvious.

References

[1] Brodtmann, T.; Litzenberger, G.: Presentation slides of "Pressegespräch VDMA Robotik + Automation", 2.3.2005.
[2] SMErobot homepage: http://www.smerobot.org
[3] Papas homepage: www.projekt-papas.de
[4] Riedl, M.; Simon, R.; Thron, M.: EDDL – Electronic Device Description Language. München, Oldenburg Industrieverlag, 2002.
[5] XML-basiertes Kommunikationsprotokoll für Industrieroboter und prozessorgestützte Peripheriegeräte, Stand 17.10.2005. Information sheet downloadable from: http://www.vdma.org/xirp
[6] UPnP Device Architecture; Version 1.0; 8.6.2000. Downloadable from the UPnP-Forum: http://www.upnp.org
[7] OWL Web Ontology Language Overview, 10.4.2004. Downloadable from W3C: http://www.w3.org/TR/owl-features/
[8] Berners-Lee, T.; Hendler, J.; Lassili, O.: The Semantic Web, Scientific American, 17.1.2001.
[9] SmartKom homepage: http://www.smartkom.org/
[10] Torge, S.; Hying, C.: Realizing Complex User Wishes With a Function Planning Module. In: SmartKom: Foundations of Multimodal Dialogue Systems. Berlin. Heidelberg. Springer Verlag, 2006.
[11] Naumann, M.; Wegener, K.; Schraft, R. D.; Lachello, L.: Robot Cell Integration by means of Application-P'n'P. In: Proceedings of ISR 2006.
[12] Homepage: www.dl.kr.org
[13] State Chart XML (SCXML): State Machine Notation for Control Abstraction 1.0, W3C Working Draft , 24.1.2006. Downloadable from W3C: http://www.w3.org/TR/scxml/
[14] Hans, M.: Eine modulare Kontrollarchitektur für den Hol- und Bringdienst von Roboterassistenten. Dissertation am IFF der Universität Stuttgart. Heimsheim. Jost-Jetter Verlag, 2005.
[15] Documentation and download: http://pellet.owldl.com/
[16] La Valle, S. M.: Planning Algorithms. University of Illinois, 2006.